

01 | 重新认识C++：生命周期和编程范式

2020-05-06 罗剑锋

罗剑锋的C++实战笔记

[进入课程 >](#)




讲述：Chrono

时长 13:20 大小 12.22M



你好，我是 Chrono。

今天是专栏的第一节正式课。我想，既然你选择了这个专栏，你就应该已经对 C++ 有所了解了，而且有过一些开发经验，甚至还掌握了一两种其他的语言。

苏轼有诗云：“不识庐山真面目，只缘身在此山中。”学习 C++ 很容易被纷繁复杂的语法细节所吸引、迷惑，所以，我决定从“**生命周期**”和“**编程范式**”这两个不太常见的角度来“剖析”一下 C++，站在一个更高的层次上审视这门“历久弥新”的编程语言，帮  清楚 C++ 最本质的东西。

这样，今后在写程序的时候，你也会有全局观或者说是大局观，更能从整体上把握程序架构，而不会迷失在那些琐碎的细枝末节里。

现在，我们先来了解下 C++ 的生命周期。

C++ 程序的生命周期

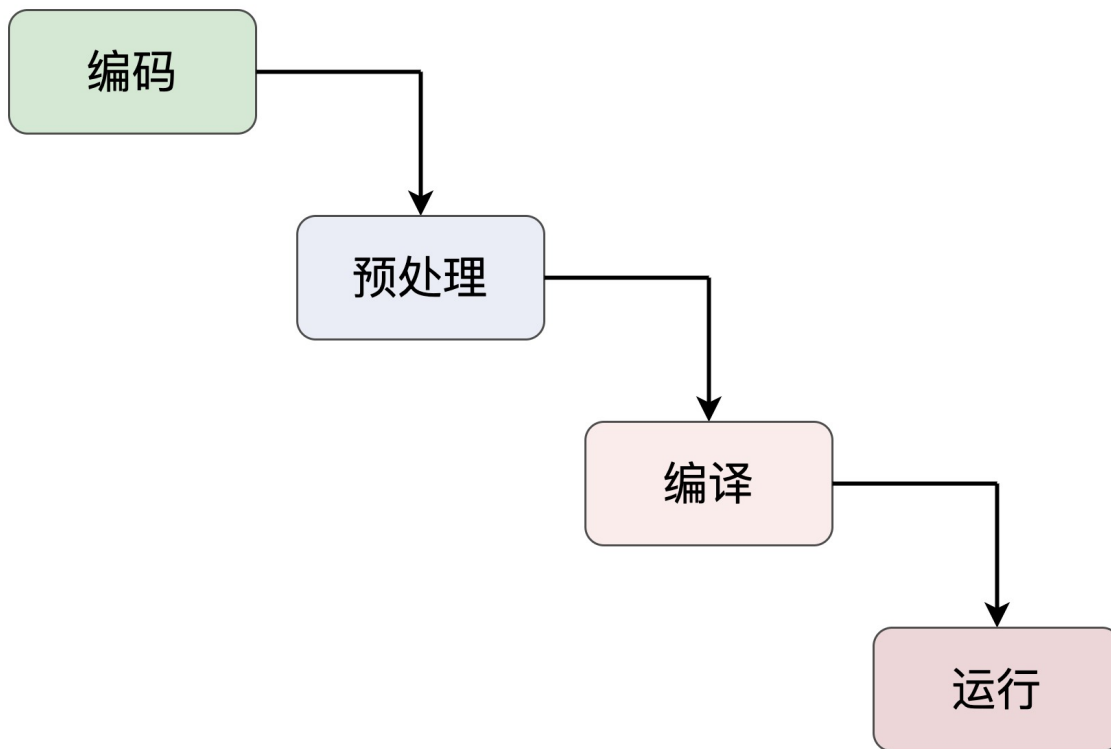
如果你学过一点软件工程的知识，就一定知道“瀑布模型”，它定义了软件或者是项目的生命周期——从需求分析开始，经过设计、开发、测试等阶段，直到最终交付给用户。

“瀑布模型”把软件的生命周期分成了多个阶段，每个阶段之间分工明确，相互独立，而且有严格的先后次序，是一个经典的开发模型。虽然它已经不再适合瞬息万变的互联网产业了，但仍然有许多值得借鉴和参考的地方。

那么，说了半天，“瀑布模型”跟 C++ 程序有什么关系呢？

其实，从软件工程的视角来看，一个 C++ 程序的生命周期也是“瀑布”形态的，也可以划分出几个明确的阶段，阶段之间顺序衔接，使用类似的方法，就可以更好地理解 C++ 程序的运行机制，帮助我们写出更好的代码。

不过，因为 C++ 程序本身就已经处在“开发”阶段了，所以不会有“需求分析”“设计”这样的写文档过程。所以，一个 C++ 程序从“诞生”到“消亡”，要经历这么几个阶段：**编码（Coding）、预处理（Pre-processing）、编译（Compiling）和运行（Running）**。



C++程序的四个阶段

C++ 程序的四个阶段

编码应该是你很熟悉的一个阶段了，这也是我们“明面”上的开发任务最集中的地方。

在这个阶段，我们的主要工作就是在编辑器里“敲代码”：定义变量，写语句，实现各种数据结构、函数和类。

编码阶段是 C++ 程序生命周期的起点，也是最重要的阶段，是后续阶段的基础，直接决定了 C++ 程序的“生存质量”。

显然，在编码阶段，我们必须依据一些规范，不能“胡写一气”，**最基本的要求是遵循语言规范和设计文档，再高级一点的话，还有代码规范、注释规范、设计模式、编程惯用法，等等。**现在市面上绝大部分的资料都是在教授这个阶段的知识，在专栏后面，我也会重点讲一讲我在这方面的一些经验积累。

那么，编码阶段之后是什么呢？

可能对你来说稍微有点陌生，这个阶段叫**预处理**。

所谓的预处理，其实是相对于下一个阶段“编译”而言的，在编译之前，预先处理一下源代码，既有点像是编码，又有点像是编译，是一个中间阶段。

预处理是 C/C++ 程序独有的阶段，其他编程语言都没有，这也算是 C/C++ 语言的一个特色了。

在这个阶段，发挥作用的是预处理器（Pre-processor）。它的输入是编码阶段产生的源码文件，输出是经过“预处理”的源码文件。“预处理”的目的是文字替换，用到的就是我们熟悉的各种预处理指令，比如 #include、#define、#if 等，实现“**预处理编程**”。这部分内容，我后面还会展开讲。

不过，你要注意的，它们都以符号“#”开头，虽然是 C++ 程序的一部分，但严格来说不属于 C++ 语言的范畴，因为它走的是预处理器。

在预处理之后，C++ 程序就进入了**编译**阶段，更准确地说，应该是“编译”和“链接（Linking）”。简单起见，我统一称之为“编译”。

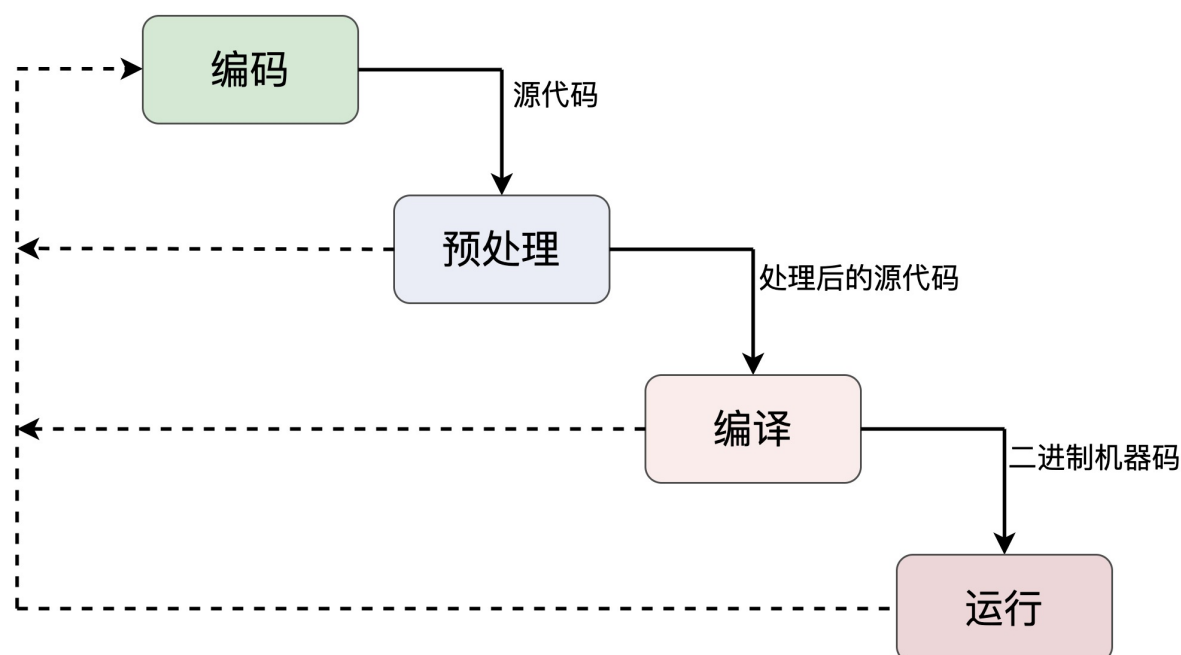
在编译阶段，C++ 程序——也就是经过预处理的源码——要经过编译器和链接器的“锤炼”，生成可以在计算机上运行的二进制机器码。这里面的讲究是最多的，也是最复杂的，C++ 编译器要分词、语法解析、生成目标码，并尽可能地去优化。

在编译的过程中，编译器还会根据 C++ 语法规则检查程序的语法、语义是否正确，发现错误就会产生“编译失败”。这就是最基本的 C++ “静态检查”。

在处理源码时，由于编译器是依据 C++ 语法检查各种类型、函数的定义，所以，在这个阶段，我们就能够以编译器为目标进行编程，有意识地控制编译器的行为。这里有个新名词，叫“**模板元编程**”。不过，“模板元编程”比较复杂，不太好理解，属于比较高级的用法，稍后我会再简单讲一下。

编译阶段之后，有了可执行文件，C++ 程序就可以跑起来了，进入**运行**阶段。这个时候，“静态的程序”被载入内存，由 CPU 逐条语句执行，就形成了“动态的进程”。

运行阶段也是我们最熟悉的了。在这个阶段，我们常做的是 GDB 调试、日志追踪、性能分析等，然后收集动态的数据、调整设计思路，再返回编码阶段，重走这个“瀑布模型”，实现“螺旋上升式”的开发。



好了，梳理清楚了 C++ 程序生命周期的四个阶段，你可以看到，这和软件工程里的“瀑布模型”很相似，这些阶段也是职责明确的，前一个阶段的输出作为后一个阶段的输入，而且每个阶段都有自己的工作特点，我们可以有针对性地去编程开发。

还有，别忘了软件工程里的“蝴蝶效应”“混沌理论”，大概意思是：一个 Bug 在越早的阶段发现并解决，它的价值就越高；一个 Bug 在越晚的阶段发现并解决，它的成本就越高。

所以，依据这个生命周期模型，**我们应该在“编码”“预处理”“编译”这前面三个阶段多下功夫，消灭 Bug，优化代码，尽量不要让 Bug 在“运行”阶段才暴露出来，也就是所谓的“把问题扼杀在萌芽期”。**

C++ 语言的编程范式

说完了 C++ 程序的生命周期，再来看看 C++ 的编程范式 (Paradigm)。

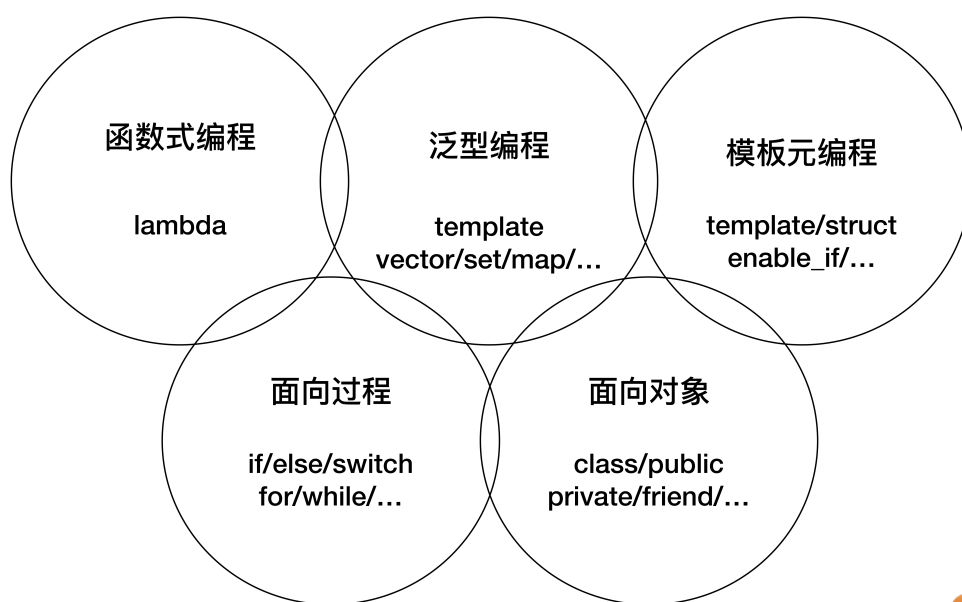
什么是编程范式呢？

关于这个概念，没有特别权威的定义，我给一个比较通俗的解释：“编程范式”是一种“方法论”，就是指导你编写代码的一些思路、规则、习惯、定式和常用语。

编程范式和编程语言不同，有的范式只能用于少数特定的语言，有的范式却适用于大多数语言；有的语言可能只支持一种范式，有的语言却可能支持多种范式。

那么，你一定知道或者听说过，C++ 是一种**多范式**的编程语言。具体来说，现代 C++（11/14 以后）支持“面向过程”“面向对象”“泛型”“模板元”“函数式”这五种主要的编程范式。

其中，“面向过程”“面向对象”是基础，支撑着后三种范式。我画了一个“五环图”，圆环重叠表示有的语言特性会同时应用于多种范式，可以帮你理解它们的关系。



极客时间

C++编程范式的“五环图”

接下来，我就和你详细说说这五种编程范式。

C++ 语言的五种范式

面向过程是 C++ 里最基本的一种编程范式。它的核心思想是“命令”，通常就是顺序执行的语句、子程序（函数），把任务分解成若干个步骤去执行，最终达成目标。

面向过程体现在 C++ 中，就是源自它的前身——C 语言的那部分，比如变量声明、表达式、分支 / 循环 / 跳转语句，等等。

面向对象是 C++ 里另一个基本的编程范式。它的核心思想是“抽象”和“封装”，倡导的是把任务分解成一些高内聚低耦合的对象，这些对象互相通信协作来完成任务。它强调对象之间的关系和接口，而不是完成任务的具体步骤。

在 C++ 里，面向对象范式包括 class、public、private、virtual、this 等类相关的关键字，还有构造函数、析构函数、友元函数等概念。

泛型编程是自 STL（标准模板库）纳入到 C++ 标准以后才逐渐流行起来的新范式，核心思想是“一切皆为类型”，或者说是“参数化类型”“类型擦除”，使用模板而不是继承的方式来复用代码，所以运行效率更高，代码也更简洁。

在 C++ 里，泛型的基础就是 template 关键字，然后是庞大而复杂的标准库，里面有各种泛型容器和算法，比如 vector、map、sort，等等。

与“泛型编程”很类似的是**模板元编程**，这个词听起来好像很新，其实也有十多年的历史了，不过相对于前三个范式来说，确实“资历浅”。它的核心思想是“类型运算”，操作的数据是编译时可见的“类型”，所以也比较特殊，代码只能由编译器执行，而不能被运行时的 CPU 执行。

在讲编译阶段的时候我也说了，模板元编程是一种高级、复杂的技术，C++ 语言对它的支持也比较少，更多的是以库的方式来使用，比如 type_traits、enable_if 等。

最后一个**函数式**，它几乎和“面向过程”一样古老，但却直到近些年才走入主流编程界的视野。所谓的“函数式”并不是 C++ 里写成函数的子程序，而是数学意义上、无副作用的函数，核心思想是“一切皆可调用”，通过一系列连续或者嵌套的函数调用实现对数据的处理。

函数式早在 C++98 时就有少量的尝试 (bind1st/bind2nd 等函数对象)，但直到 C++11 引入了 Lambda 表达式，它才真正获得了可与其他范式并驾齐驱的地位。

好了，介绍完了这五种编程范式，你可以看到，它们基本覆盖了 C++ 语言和标准库的各个成分，彼此之间虽然有重叠，但在理念、关键字、实现机制、运行阶段等方面的差异还是非常大的。

这就好像是五种秉性不同的“真气”，在 C++ 语言里必须要有相当“浑厚”的内力才能把它们压制、收服、炼化，否则的话，一旦运用不当，就很容易“精神分裂”“走火入魔”。

说得具体一点，就是要认识、理解这些范式的优势和劣势，在程序里适当混用，取长补短才是“王道”。

说到这儿，你肯定很关心，该选择哪种编程范式呢？

拿我自己来说，我的出发点是“**尽量让周围的人都能看懂代码**”，所以常用的范式是“过程 + 对象 + 泛型”，再加上少量的“函数式”，慎用“模板元”。

对于你来说，我建议根据自己的实际工作需求来决定。

我个人觉得，**面向过程和面向对象是最基本的范式，是 C++ 的基础，无论如何都是必须要掌握的**，而后三种范式的学习难度就大一些。

如果是开发直接面对用户的普通应用 (Application)，那么你可以再研究一下“泛型”和“函数式”，就基本可以解决 90% 的开发问题了；如果是开发面向程序员的库 (Library)，那么你就有必要深入了解“泛型”和“模板元”，优化库的接口和运行效率。

当然，还有一种情况：如果你愿意挑战“最强大脑”，那么，“模板元编程”就绝对是你的不二选择（笑）。

小结

今天是开篇第一课，我带你从“生命周期”和“编程范式”这两个特别的角度深度“透视”了一下 C++，做个简单小结：

1. C++ 程序的生命周期包括编码、预处理、编译、运行四个阶段，它们都有各自的特点；
2. 虽然我们只写了一个 C++ 程序，但里面的代码可能会运行在不同的阶段，分别由预处理器、编译器和 CPU 执行；
3. C++ 支持面向过程、面向对象、泛型、模板元、函数式共五种主要的编程范式；
4. 在 C++ 里可以“无缝”混用多范式编程，但因为范式的差异比较大，必须小心谨慎，避免导致混乱。

课下作业

最后是课下作业时间，给你留两个思考题：

1. 你是怎么理解 C++ 程序的生命周期和编程范式的？
2. 试着从程序的生命周期和编程范式的角度，把 C++ 和其他语言（例如 Java、Python）做个比较，说说 C++ 的优点和缺点分别是什么。

欢迎你在留言区写下你的思考和答案，如果觉得对你有所帮助，也欢迎把今天的内容分享给你的朋友，我们下节课见。

课外小贴士

1. “范式”这个词不仅存在于编程领域，其他领域也有，比如数据库里就有第一范式、第二范式、BCNF 范式。
2. C++20 的新特性“range”引入了管道符“|”，函数式编程就可以发挥出更大的作用。
3. 不太严格地讲，预处理阶段的“预处理元编程”也可以算作是一个编程范式，但它的局限性比较大，功能略弱，还不能和其他范式相提并论。

上一篇 课前准备 | 搭建实验环境

下一篇 02 | 编码阶段能做什么：秀出好的code style

精选留言 (25)

写留言



八曲长弓

2020-05-06

以前看过罗老师的《BOOST程序库完全开发指南》有不少收获，极客上看见罗老师C++课程毫不犹豫就买了，希望跟罗老师学习更多。

展开

作者回复: 感谢老读者捧场，有Boost的问题也可以在这里随时问。



6



忆水寒

2020-05-06

- 1、编程范式其实用的最多的就是面向过程、面向对象、类模板和泛型编程。
- 2、C++的优点是运行效率高，毕竟比较靠近底层硬件了。C++的缺点就是面向对象的不彻底，多重继承确实比较混乱难理解。

展开

作者回复: 说的很好。

对于第二点，C++不是不彻底，而是不存粹，毕竟它是面向对象的探路先锋，走了一些弯路，给后来的java等语言蹚了道。

其实如果你愿意，完全可以在C++里按照Java的风格来写面向对象，这就是C++给予我们的自由。



5



Carlos

2020-05-06

回答老师的问题并提出我自己的问题:

1. 生命周期我比较熟悉, 这个过程可以用之前对编程范式了解不多, 今天头一次听说, 我觉得这个好像更是一种 "解决问题的习惯". 请问老师网上的 google c++ style guide 算不算是一直种范式? ...

展开 ▾

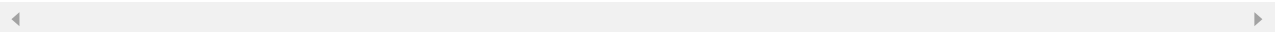
作者回复: 回答的很认真, 表扬。

1.编程范式是一种方法论, 指导如何编程, 偏重语法规则。Google style guide是编码风格指南, 还不算是范式。可以结合课程正文再理解一下。

2.比较的挺好, 一个是解释型语言, 一个是编译型语言。

3.C++的语法要素中英文翻译都比较规范了, 如果纯英文有时候会有点别扭, 有不清楚的就问吧, 我给解答。

4.这个名字来自于超任的《chrono trigger》, 25年前的老游戏。



💬 1

👍 3



Woturbo

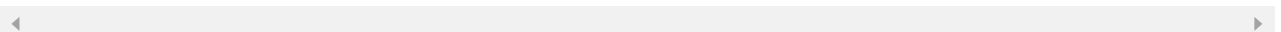
2020-05-09

1. 生命周期是值得整个C++程序的开发运行过程, 除编码外都是由编译器完成。像预编译、编译、链接的过程实际知道原理即可, 达到出现问题的时候能够解决。编程范式就是编程思想 (方法路), c++复杂的根源, 包含了五种。功能是足够强大, 但如果不合理使用, 一味的去 "炫技", 会不利于开发。核心思路是以可读性和可维护性为第一原则。...

展开 ▾

作者回复: 总结的很好。

在C++里, 所谓的鸭子对象, 其实就是泛型。



💬

👍 1



斐

2020-05-09

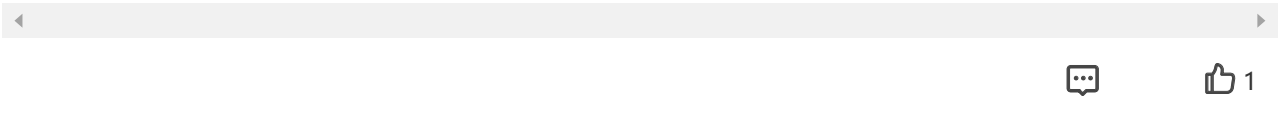
罗老师, 面向对象和基于对象有什么区别?

展开 ▾

作者回复: 这两个广义上都属于面向对象, 严格来说也算是有区别。

前者强调统一用对象建模, 多应用设计模式, 对象关系复杂。

而后者则相当于C with class, 只把class当成struct来封装数据, 继承、多态等高级特性用的比较少。



拉普达

2020-05-07

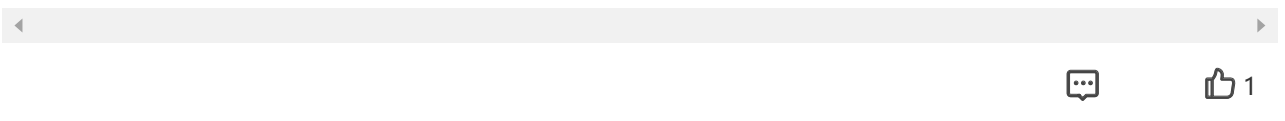
1、C++生命周期中的预处理和编译是其特有的环节, 编译执行是其高效的基础, 相比而言java、python等解释执行的语言更有优势。但正是出于效率的考虑, C++的编译执行给程序员提供了足够的灵活性, 但也提供了无限出错的可能, 这也是其难学的原因。另外, 关于C++的预处理和编译, 希望老师在后续课程中多花些时间讲解一下, 之前每次遇到Link Error, 都不知道怎么处理。...

展开 ∨

作者回复:

1.编译链接出错确实是很多C++程序员最头痛的问题, 报错信息一大堆, 很不友好。我的方法是去搜索引擎里搜一下错误文字, 通常都能够找到答案或者提示。

2.理解的很对, C++的应用水平取决于团体的整体水平, 但如果总停留在低水平既不利于项目也不利于自己, 可以通过培训、讨论来提高, 这样C++就会越用越爽。



乐生 ...

2020-05-11

罗老师, 您好, 主要目前的工作全都是c++11新特性写的代码, 我之前工作两年都是98标准, 刚入职两天。对新特性的代码读的不太懂, 想通过您的课程, 快速去理解代码, 但是想知道 如何快速的去理解新特性的代码, 现在就是边搜索。边去理解。但是感觉很困难, 搜到的例子很简单易懂。但是结合项目。就觉得别人写的很深奥。

展开 ∨



土豆

2020-05-10

1、C++程序的生命周期, 分为: 编辑代码 -> 预处理器进行相关宏定义, typedef等替换

-》编译器编译成汇编代码，汇编器生成目标obj -》ld连接器进行相应库的链接 -》生成的可执行文件其实就是这些obj的打包文件，如果是静态链接的话则还包含静态库的内容，如果是动态库则包含执行动态库的路径 -》执行起来成为一个进程 -》进程结束意味着该C++程序的生命周期结束了； ...

展开 ▾

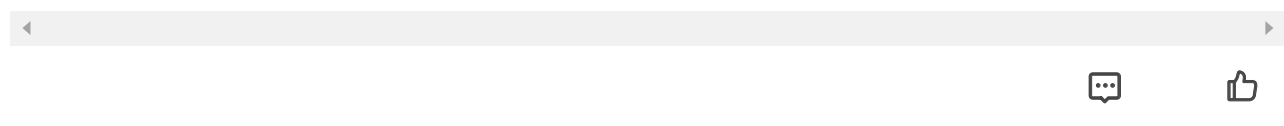
作者回复: 总结的很好，有几个小错误要说一下。

1.typedef是在编译阶段，而不是预处理阶段，这个后面讲预处理、编译的时候你就清楚了。

2.直接编译成机器码，但这并不是跨平台的代码，而是源码可以跨平台，由不同的编译器编译成对应平台的二进制代码。

Java等语言的字节码由于运行在虚拟机上，它才是跨平台的代码。

3.C++编译慢还有一个原因，就是语法复杂，还有编译优化等等。



阿鼎

2020-05-10

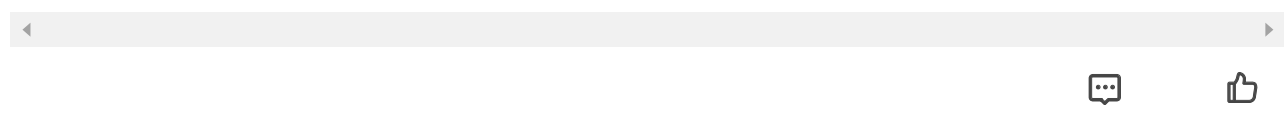
由于项目代码比较老，现在很多开源库都是11的。请问老师可否将c++11的代码，能否逆向转化成c++03的？对于老工程，语言生态已经停滞更新，如何应对？

展开 ▾

作者回复:

1.可行，但难度会比较大，C++11里的很多新特性在98/03里只能模拟实现，甚至无法模拟。

2.这种情况就要重构了，不破不立，不要总是背着历史的包袱。像gcc，以前一直用C，后来终于下狠心，转成了C++，转完以后就爽了。



cww

2020-05-08

生命周期就像想法到实现的过程。从编码到运行是单个cpp程序的生命周期，同时也是一个大的cpp工程的生命周期。

编程范式是实现想法的设计方法，有道是条条大路通罗马，编程范式更像是通往罗马（实现设计）的一条道路。

展开 ▾

作者回复: 说的很好, 理解了这两个概念, 我们才能把握C++的本质, 把C++的特性分类分治来学习。



The rustic

2020-05-08

老师, 请教个问题, 我目前坐容器相关得工作, 但是开发很少。一直认为程序员开发得基本功必须扎实, 所以又过来看看久违的c++。老师觉得我应该换个工作提高开发水平, 还是其他

作者回复: 不知道你说的容器是不是docker, 它用的是go。

如果是这样的话, 我觉得现在微服务、service mesh、云还是很热门的, 可以继续努力。

但学习C++可以让你打好基础, 理解底层计算机的运行机制, 反过来让你在上层更好地工作。

而且C++也很容易编写各种底层模块给上层调用, 做混合编程里最核心最要求性能的那部分。

2



哈珀朋友

2020-05-07

感兴趣的两个方向图形学和音视频都是C++逃不掉, C++性能和效率还是高啊

作者回复: 这两个方向都是C++擅长的领域, 可惜我没往这方面发展。



wuwei

2020-05-07

学过一点点cpp。

生命周期: 了解到cpp的从写到运行的机制, 不再是黑盒子;

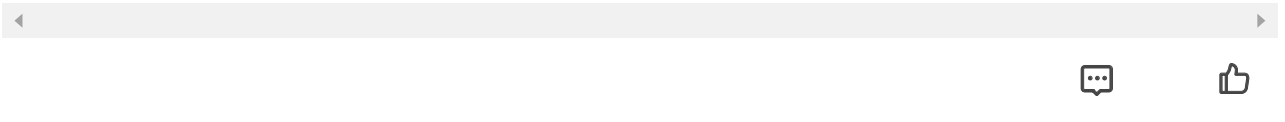
编程范式: 面向过程, 面向对象, 泛化编程, 函数式编程, 模版元这些范式让我明白, 范式有自己的特点和优势, 清楚的掌握他们, 将他们作为工具充分利用, 应该可以避免混淆; ...

展开 ∨

作者回复: 总结的很好。

你提的这些特性，多线程复杂但有用，重载、多态学起来容易但有很多陷阱，智能指针是好东西，后面我会专门讲，它和容器都可以用来管理动态内存。

我的建议是先学简单的特性，复杂易错的特性少用，有经验了再扩大使用范围。



jxon-H

2020-05-07

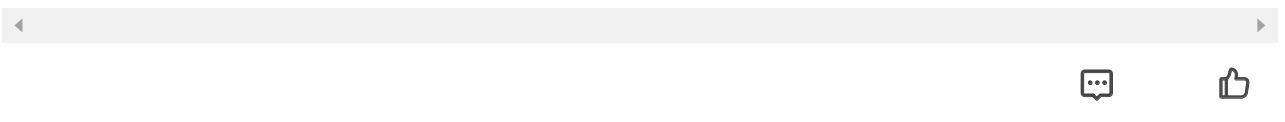
我的C++一直没学懂没学好。希望能跟上罗老师的步伐，填补我知识结构的空缺。

罗老师把C++的生命周期与瀑布模型进行类比，形象地把C++的编程过程进行剖析，通过在生命周期四个过程的来回改进，实现程序的螺旋式上升，巧妙地说清了软件开发是个不断迭代的过程。...

展开 ▾

作者回复: 说得很好。

生命周期这个其实是为了方便大家理解C++程序，把它分解成几个阶段，有的C++特性只在某个阶段起作用，这样就会认识的比较清楚。



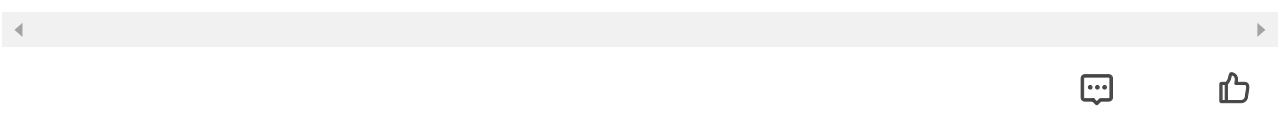
文若

2020-05-07

能把问题通俗易懂的讲出来,才见功夫,老师厉害!不知道后续课程里面有Boost相关的知识点没有,想多了解Boost相关?

展开 ▾

作者回复: Boost太大，不太好讲，下次如果有机会再说吧，这次的主题是C++本身。



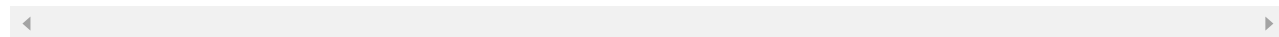
natural

2020-05-07

罗老师考虑出视频教程么？

展开 ▾

作者回复: 暂时还没有想法, 我个人觉得能用文字和音频解决最好, 视频教程花的精力比较多, 抱歉啊。



💬 1



chen_julio

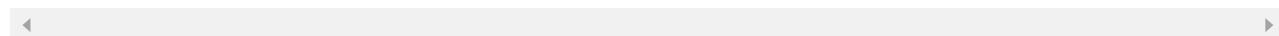
2020-05-07

模版元简直变态, 各种黑魔法

展开 ▾

作者回复: 在C++98时代模板元编程特别复杂, 到了C++11/14就好一些了, 因为有了using别名, 可以在编译阶段写类型的赋值语句。

不过模板元编程还是属于高级技巧, 应用的场合比较少, 所以这次的专栏就暂时不讲。



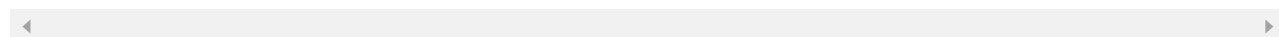
luoyayun361

2020-05-06

准备上车了

展开 ▾

作者回复: 共同努力。

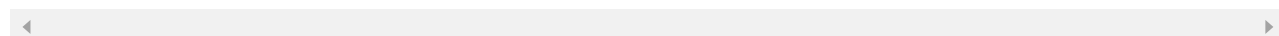


Sun

2020-05-06

老师好, 在编译和链接是不是还有一个汇编阶段, 汇编器将汇编代码转换成目标代码

作者回复: 精确地说是的, 但我这里把由源码转换到二进制的过程统一称为编译阶段, 省事了。



少年

2020-05-06

老师的思路确实给徘徊在入门与放弃C++的人提供了一条新路，听完很有感触。生命周期的阶段划分，预处理单独成段？为啥不叫编译预处理？

展开 ▾

作者回复: 预处理是C语言的传统，就是文本替换，不涉及语法，英文术语就叫pre-process，用原文来理解吧。

